# MIRI Software Documentation

***Release 2.5evxx***

**MIRI Software Team**

December 18, 2015

**1 Installation** **3**
1.1 MIRI Software 2.5evxx Installation Instructions . . . . . 3

**2 Release Notes** **5**
2.1 MIRI top level software release notes (`miri`) . . . . . 5
2.2 MIRI miritools release notes (`miri.miritools`) . . . . . 5
2.3 MIRI imaging release notes (`miri.imaging`) . . . . . 5
2.4 MIRI coronography release notes (`miri.coronography`) . . . . . 5
2.5 MIRI spectroscopy release notes (`miri.spectroscopy`) . . . . . 5
2.6 MIRI simulators release notes (`miri.simulators`) . . . . . 6
2.7 MIRI stpipeline release notes (`miri.stpipeline`) . . . . . 6

**3 Package Reference** **7**
3.1 Common tools for MIRI Software (`miri.miritools`) . . . . . 7
3.2 MIRI Imaging Package (`miri.imaging`) . . . . . 23
3.3 MIRI Coronography Package (`miri.coronography`) . . . . . 23
3.4 MIRI Spectroscopy Package (`miri.spectroscopy`) . . . . . 23
3.5 MIRI Simulator Software (`miri.simulators`) . . . . . 24
3.6 MIRI STScI Pipeline Support (`miri.stpipeline`) . . . . . 44

**4 Appendix - Installation Utilities** **45**
4.1 Installation utilities (`miri.setup_utils`) . . . . . 45

**Python Module Index** **47**

**Index** **49**

**Release** 2.5evxx

**Date** December 18, 2015

CHAPTER

ONE

# INSTALLATION

## 1.1 MIRI Software 2.5evxx Installation Instructions

**Release** 2.5evxx

**Date** December 18, 2015

**Author** Steven Beard <steven.beard@stfc.ac.uk>

### 1.1.1 Notes

- The source file for this page is available at: https:https://aeon.stsci.edu/ssb/svn/jwst/trunk/teams/miri/doc/source/install.rst
- The recommended way of installing this software is using the MIRICLE script described at: http://miri.ster.kuleuven.be/bin/view/Internal/MiricleSoftwareSystem

  The MIRICLE script can be downloaded from http://miri.ster.kuleuven.be/pub/Internal/MiricleSoftwareSystem/MIRICLE_install.
- MIRICLE is designed for operating systems which understand the bash script. For installation on Windows workstations, or anything else without bash, the following MIRI project wiki pages provide alternative instructions:
  - Installation instructions for Python and stsci_python may be found at: http://miri.ster.kuleuven.be/bin/view/Internal/Software/PythonInstall
  - Information about the configuration of the MIRI software may be found at: http://miri.ster.kuleuven.be/bin/view/Internal/Software/PythonCommonCodeRepository
  - Information on the installation and use of SCASim may be found at: http://miri.ster.kuleuven.be/bin/view/Internal/Software/ScaSim

#### Building the MIRI Software

The software is built by executing the command

python setup.py install

from the top level teams/miri directory.

#### Test the MIRI software installation

The "teams/miri/miri_installation_check.py" script can be run to check that all the necessary modules have been installed.

The “teams/miri/miri_run_tests.py” script can be run to execute all the MIRI module unit tests.

CHAPTER

TWO

# RELEASE NOTES

## 2.1 MIRI top level software release notes (`miri`)

**Release** 2.5evxx

**Date** December 18, 2015

Release notes for the whole MIRI software suite are written here.

See the release notes for each individual package for details.

## 2.2 MIRI miritools release notes (`miri.miritools`)

**Release** 2.5evxx

**Date** December 18, 2015

Release notes for the miritools package are written here.

## 2.3 MIRI imaging release notes (`miri.imaging`)

**Release** 2.5evxx

**Date** December 18, 2015

Release notes for the MIRI imaging package are written here.

## 2.4 MIRI coronography release notes (`miri.coronography`)

**Release** 2.5evxx

**Date** December 18, 2015

Release notes for the MIRI coronography package are written here.

## 2.5 MIRI spectroscopy release notes (`miri.spectroscopy`)

**Release** 2.5evxx

**Date** December 18, 2015

Release notes for the MIRI spectroscopy package are written here.

## 2.6 MIRI simulators release notes (`miri.simulators`)

**Release** 2.5evxx

**Date** December 18, 2015

### 2.6.1 General notes

General release notes for all MIRI simulator packages are written here.

Release notes for specific simulator packages are written below.

### 2.6.2 Module specific release notes

## 2.7 MIRI stpipeline release notes (`miri.stpipeline`)

**Release** 2.5evxx

**Date** December 18, 2015

Release notes for miri STScI pipeline support utilities are written here.

CHAPTER

THREE

# PACKAGE REFERENCE

## 3.1 Common tools for MIRI Software (`miri.miritools`)

**Release** 2.5evxx

**Date** December 18, 2015

### 3.1.1 LICENCE

#### Copyright

Copyright (c) 2010-2015 The JWST MIRI European Consortium software team. All rights reserved.

The miritools software has been developed by the MIRI EC software team as part of the JWST/MIRI consortium, which includes the following organisations: Ames Research Center, USA; Netherlands Foundation for Research in Astronomy; CEA Service d'Astrophysique, Saclay, France; Centre Spatial de Liege, Belgium; Consejo Superior de Investigacones Cientificas, Spain; Danish Space Research Institute; Dublin Institute for Advanced Studies, Ireland; EADS Astrium, Ltd., European Space Agency, Netherlands; UK; Institute d'Astrophysique Spatiale, France; Instituto Nacional de Tecnica Aerospacial, Spain; Institute of Astronomy, Zurich, Switzerland; Jet Propulsion Laboratory, USA; Laboratoire d'Astrophysique de Marseille (LAM), France; Lockheed Advanced Technology Center, USA; Max-Planck-Insitut fur Astronomie (MPIA), Heidelberg, Germany; Observatoire de Paris, France; Observatory of Geneva, Switzerland; Paul Scherrer Institut, Switzerland; Physikalishes Institut, Bern, Switzerland; Raytheon Vision Systems, USA; Rutherford Appleton Laboratory (RAL), UK; Space Telescope Science Institute, USA; Toegepast-Natuurwetenschappelijk Ondeszoek (TNOTPD), Netherlands; UK Astronomy Technology Centre (UKATC); University College, London, UK; University of Amsterdam, Netherlands; University of Arizona, USA; University of Cardiff, UK; University of Cologne, Germany; University of Groningen, Netherlands; University of Leicester, UK; University of Leiden, Netherlands; University of Leuven, Belgium; University of Stockholm, Sweden, Utah State University USA.

#### Terms and Conditions of Use

This software may be used and copied free of charge only for non-commercial research purposes. All copies of this software must contain this copyright statement and disclaimer. The MIRI consortium must be acknowledged in any publications arising from use of this software. If you make modifications to this software, you must clearly mark the software as having been changed and you must also retain this copyright and disclaimer.

Where this software uses facilities developed by other members of the MIRI consortium (e.g. its use of stsci-python) it is also bound by the licences issued with those facilities (see the LICENCE files contained in other parts of the JWST repository).

#### Disclaimer

This software is available "as is", without warranty of any kind, either expressed or implied, including the implied warranties of merchantability and fitness for a specific purpose. By using this software you are assuming all risks and costs. In no event is the MIRI EC software team or the MIRI consortium liable for any damages or losses that might result from the use of this software.

Any calibration files distributed with this software are for testing purposes and do not describe the status of the MIRI instrument or the quality of the data expected from MIRI.

### 3.1.2 Introduction

This document describes the implementation details for the MIRI general purpose tools and utilities package (miritools). Tools may be found within the top level miritools module, or they may be categorized into subpackages, such as miritools.dataproduct.

### 3.1.3 Modules

General purpose miritools modules may be found in the miritools/lib directory

#### File location utilities (`miri.miritools.filesearching`)

##### Description

This module contains general purpose functions for locating files.

##### Objects

##### Functions

#### Plotting module (`miri.miritools.miriplot`)

##### Description

This module contains general purpose functions for plotting MIRI data. In fact, the functions can be used for plotting any numpy array data which matches the function parameters described. It is designed to be used as a toolbox by other MIRI software. The "plot" function may be run directly from a Python shell.

Note: The plotting functions use matplotlib. It is important to use the close() function to tidy up and remove old figures after making a series of plots. Failure to tidy up regularly may result in old figures popping back into view on the next call to showplot().

##### Functions

Unit tests corresponding to these modules may be found in the miritools/tests directory.

### 3.1.4 Scripts

General purpose miritools scripts may be found in the miritools/scripts directory.

At the moment there are no top level scripts. The scripts associated with each subpackage may be found in the miritools/subpackage/scripts directory.

### 3.1.5 Data

General purpose miritools data may be found in the miritools/data directory.

At the moment there are no top level data. The data associated with each subpackage may be found in the miritools/subpackage/data directory.

### 3.1.6 Subpackage - dataproduct

The dataproduct modules may be found in the miritools/dataproduct/lib directory.

#### MIRI Base Data Module (`dataproduct.miri_model_base`)

##### Description

This module contains the MiriDataModel class, which is the base class for all the MIRI data models. The class builds on the functions provided by the STScI data model class DataModel and adds extra functions specific to MIRI. Convenience functions are also included which implement some of the features of the old MIRI data products.

All MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

##### Objects

##### Functions

None

##### Data formats

TBD.

#### MIRI Measured Data Module (`dataproduct.miri_measured_model`)

##### Description

This module contains the MiriMeasuredModel, MiriRampModel and MiriSlopeModel, classes, which describe generic MIRI data models containing a data array, error array and data quality array. (The ramp model separates the pixel and group data quality into two arrays.) The classes also include functions to support simple arithmetic operations, mainly for interactive use.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Bad Pixel Data (`dataproduct.miri_badpixel_model`)

#### Description

This module contains the MiriBadPixelMaskModel class, which describes a MIRI bad pixel mask.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Global Data

mask_reference_flags - Defines the MIRI/JWST pixel mask reference flags

#### Data formats

TBD.

### Dark Reference Data (`dataproduct.miri_dark_reference_model`)

#### Description

This module contains the MiriDarkReferenceModel class, which describes MIRI dark reference data.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

WARNING: File containing dark reference data tend to be very large. Use slice operations to read only the relevant parts of the data structure. See the memory management part of the jwst_lib.models documentation for details.

#### Objects

#### Functions

None

#### Global Data

dark_reference_flags - Defines the MIRI/JWST dark reference flags

#### Data formats

TBD.

### Distortion Coefficients (`dataproduct.miri_distortion_models`)

#### Description

This module contains the MiriImagingDistortionModel class and related classes, which describe MIRI distortion coefficients.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Droop Correction Coefficients (`dataproduct.miri_droop_model`)

#### Description

This module contains the MiriDroopModel class, which describes MIRI droop correction coefficients.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Flat-Field Data (`dataproduct.miri_flatfield_model`)

#### Description

This module contains the MiriFlatfieldModel class, which describes MIRI flat-field data. The same model is used for pixel flats, fringe flats and sky flats (differing only in the data type stored in the metadata).

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Global Data

flat_reference_flags - Defines the MIRI/JWST flat-field reference flags

#### Data formats

TBD.

### Flux Conversion Data (`dataproduct.miri_fluxconversion_models`)

#### Description

This module contains the MiriFluxconversionModel, MiriImagingFluxconversionModel and MiriSpectroscopyFluxconversionModel classes, which describe MIRI flux conversion factors. The imaging and spectroscopy models differ in the way the wavelength dependency is parameterized.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Detector Gain Data (`dataproduct.miri_gain_model`)

#### Description

This module contains the MiriGainModel class, which describes MIRI detector amplifier gains.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Jump Detection Coefficients (`dataproduct.miri_jump_model`)

#### Description

This module contains the MiriJumpModel class, which describes MIRI jump detection coefficients.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Last Frame Correction Model (`dataproduct.miri_lastframe_model`)

#### Description

This module contains the MiriLastFrameModel class, which describes MIRI last frame correction coefficients.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Latent Decay Model (`dataproduct.miri_latent_model`)

#### Description

This module contains the MiriLatentDecay and MiriLatentTable classes, which describe the MIRI detector latent decay model.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Linearity Data (`dataproduct.miri_linearity_model`)

#### Description

This module contains the MiriLinearityModel class, which describes MIRI nonlinearity coefficents.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Global Data

linearity_reference_flags - Defines the MIRI/JWST linearity reference flags

#### Data formats

TBD.

### Flux Conversion Data (`dataproduct.miri_photmetric_models`)

#### Description

This module contains the MiriPhotometricModel classes, which describe MIRI photmetric conversion factors. It also contains the MiriPixelAreaModel classes.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Pixel Saturation Model (`dataproduct.miri_pixel_saturation_model`)

#### Description

This module contains the MiriPixelSaturationModel class, which contains pixel saturation data.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Global Data

saturation_reference_flags - Defines the MIRI/JWST saturation reference flags

#### Data formats

TBD.

### Point Spread Function Data (`dataproduct.miri_psf_models`)

#### Description

This module contains the MiriPointSpreadFunctionModel and related classes, which describe the JWST/MIRI point spread function measurements.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Detector Reset Model (`dataproduct.miri_reset_model`)

#### Description

This module contains the MiriResetModel class, which describes MIRI detector reset level

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Stray Light Model (`dataproduct.miri_straylight_model`)

#### Description

This module contains the MiriMrsStraylightModel class, which describes MIRI MRS stray light properties.

Further stray light models may be added here.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Telescope Emission Data (`dataproduct.miri_telescope_emission_model`)

#### Description

This module contains the MiriTelescopeEmissionModel class, which describes MIRI telescope emission data.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Transmission Correction (`dataproduct.miri_transmission_correction__model`)

#### Description

This module contains the MiriTransmissionCorrectionModel class, which describes transmission corrections to be applied to MIRI MRS data.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### Wavelength Correction (dataproduct.miri_wavelength_correction__model)

#### Description

This module contains the MiriWavelengthCorrectionModel class, which describes across slice wavelength corrections to be applied to MIRI MRS data.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

### MIRI simulation data definition module (dataproduct.miri_illumination_model)

#### Description

This module contains the MiriIlluminationModel class, which is designed to be used by simulators to describe an illumination pattern and create simulated JWST data.

#### Objects

#### Functions

None.

#### Data formats

TBD.

### MIRI simulation data definition module (dataproduct.miri_exposure_model)

#### Description

This module contains the MiriExposureModel class, which is designed to be used by simulators to store simulated JWST exposure data.

#### Objects

#### Functions

TBD.

#### Data formats

TBD.

### MIRI Data Model Operations Module (`dataproduct.operations`)

#### Description

This module contains the HasMask, HasData and HasDataErrAndDq classes, which provide functions for performing arithmetical operations on data structures containing onr or more data arrays or bitwise operations on data structures containing a mask array.

These classes are not meant to be used on their own. They are designed to supplement the functionality of other data model classes.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

See also the description of the dqflags.py module.

#### Objects

#### Functions

None

### MIRI Data Model Compatibility Module (`dataproduct.compatibility`)

#### Description

This module contains the WithCompatibility class, which can be used to help convert existing software using the old MIRI data products to use the new data products. The class provides some bolt-on attributes and functions which mimic some features of the old model which are not included in the MiriDataModel class.

NOTE: There are not many functions in this module. Long term use of this module is not recommended. It should be used temporarily while converting existing software. The source code shows how the old functionality can be implemented in the new data model.

#### Objects

#### Functions

None

#### MIRI Data Quality Flags Module (`dataproduct.dqflags`)

##### Description

This module contains the FlagsTable class, together with the functions for managing and combining data quality flags.

##### Objects

##### Functions

##### Global Data

master_flags - Defines the JWST pipeline master flags table

#### MIRI Data Model Plotting Module (`dataproduct.plotting`)

##### Description

This module contains the DataModelVisitor and DataModelPlotVisitor classes, which provide model-independent plotting functions. These classes implement the visitor design pattern, and are invoked by calling their visit() method with the data object to be plotted passed as a parameter.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

##### Objects

##### Functions

None

### 3.1.7 Data for dataproduct

Example data product data can be stored in the miritools/dataproduct/data directory. At the moment this directory is empty.

### 3.1.8 Scripts for dataproduct

The miritools.dataproduct scripts may be found in the miritools/dataproduct/scripts directory, which contains the following scripts.

#### General purpose scripts

### 3.1.9 Subpackage - measurements

The measurements modules may be found in the miritools/measurements/lib directory.

### Filter Utilities Module (`measurements.miri_filters`)

#### Description

The miri_filters module is used to read, store and apply a filter, which is described as transmission function which varies with wavelength. The transmission can vary from 0.0 (no flux transmitted) to 1.0 (flux unaffected).

A filter can also be used to describe the quantum efficiency of a detector, where the transmission function in this case represents the QE as a function of wavelength.

These classes depend on the MIRI data product module (`miri.miritools.dataproduct`).

#### Objects

#### Functions

#### Configuration files

A Filter object can be stored either within a FITS file or an ASCII table. A FITS file contains all the information about a Filter object whereas a ASCII file only contains the transmission as a function of wavelength.

A quantum efficiency measurement is defined in a similar file format, except the transmission column contains QE measurements.

**Naming convention** MIRI filter measurements are defined in the files data/filters/IM-VV-FXXXX.txt where "VV" is a version number and "FXXXX" is the name of the filter (e.g. "F2100W").

MIRI quantum efficiency measurements are defined in the files data/detector/qe_measurementXXX.txt where "XXX" is the focal plane module ID (e.g. "495").

**ASCII file format** ASCII files contain data in the following format. Lines starting with a '#' character are ignored. There is no header, and the file can contain two columns (where there is a measurement at a single temperature):

- Column #1: The wavelength in microns.
- Column #2: The quantum efficiency measurement at this wavelength.

or three columns (where there are measurements at room temperature and cryogenic temperature):

- Column #1: wavelength (micron) %.2f
- Column #2: transmission (%) %.5f
- Column #3: transmission (%) %.5f

---

**Note:** Column #2 is not used by `miri.miritools.measurements.filters`, it corresponds to room temperature measurements. Column #3 corresponds to cryogenic temperature measurement in MIRI data and is actually used by `miri.miritools.measurements.filters`.

---

**FITS File format** The FITS version of a MIRI filter is stored in a format prescribed by the JWST data model in jwst_lib, with the following additional header keywords:

- FILTERTY: Filter type, e.g. 'BP' for bandpass
- WAVECENT: Central wavelength for band-pass filter

- FWHM: FWHM wavelength for band-pass filter

The filter table may be found in an binary table HDU called 'FILTER'.

#### Measurement Utilities Module (`miri.measurements.miri_measurement`)

##### Description

This module contains the MiriMeasurement class, which is used to describe any set of measurements which vary in a monotonic manner with respect to one variable parameter. For example, dark current and read noise are expected to vary with detector temperature. This class stores the measurements as a lookup table from which the property at any parameter setting can be predicted by interpolating the measurements

*NOTE: It is assumed that calibration measurements will be made over the full range of operational conditions. The MiriMeasurement class will not give good predictions outside the range of measurements because it does not extrapolate.*

##### Objects

##### Functions

##### Configuration files

MIRI measured variables are defined in the files data/detector/dark_currentFPMSNxxx.txt and data/amplifier/read_noiseFPMSNxxx.txt where "SNxxx" is the ID of a particular focal plane module.

ASCII files contain data in the following format. Lines starting with a '#' character are ignored. The file contains two or more columns:

- Column #0: The environmental parameter value (e.g. temperature).
- Column #1: The corresponding variable measurement.
- Column #2: Another corresponding measurement of the same variable.
- etc...

Columns 2 onwards may be used to store separate measurements of the same parameter. For example, one file can be used to store the read noise measurements as a function of temperature for each amplifier within the same focal plane module.

FITS files are stored in the format specified by the JWST data model. The measurement data may be found in a binary table HDU called 'MEASUREMENT'.

### 3.1.10 Data for measurements

`miri.miritools.measurements` comes with some example filter and measurement data, provided as ASCII and FITS files (see `miri.miritools.measurements.miri_filters` and `miri.miritools.measurements.miri_measurement`).

### 3.1.11 Configuration Information

An example measurement properties file (for `miri.miritools.measurements.miri_measurement`) may be found in miritools/measurements/lib/example_properties.py.

### 3.1.12 Scripts for measurements

The miritools.measurements scripts may be found in the miritools/measurements/scripts directory.

#### make_filters_fits

#### make_qe_fits

#### make_measurements_fits

## 3.2 MIRI Imaging Package (`miri.imaging`)

**Release** 2.5evxx

**Date** December 18, 2015

INSERT DOCUMENTATION FOR THE MIRI IMAGING PACKAGE HERE.

## 3.3 MIRI Coronography Package (`miri.coronography`)

**Release** 2.5evxx

**Date** December 18, 2015

INSERT DOCUMENTATION ON MIRI CORONOGRAPHY PACKAGE HERE

## 3.4 MIRI Spectroscopy Package (`miri.spectroscopy`)

**Release** 2.5evxx

**Date** December 18, 2015

### 3.4.1 Introduction

This document describes the implementation details for the MIRI spectroscopy package (miri.spectroscopy).

ADD MORE INTRODUCTORY INFO HERE.

### 3.4.2 Modules

General purpose spectroscopy modules may be found in the spectroscopy/lib directory

#### MIRI Spectral Data Models (`spectroscopy.spectraldata`)

##### Description

This module contains the classes which describe generic MIRI spectroscopy data models.

MIRI data models depend on the STScI data model, found in the jwst_lib.models package.

#### Objects

#### Functions

None

#### Data formats

TBD.

Unit tests corresponding to these modules may be found in the spectroscopy/tests directory.

### 3.4.3 Scripts

General purpose spectroscopy scripts may be found in the spectroscopy/scripts directory.

At the moment there are no top level scripts. The scripts associated with each subpackage may be found in the spectroscopy/subpackage/scripts directory.

### 3.4.4 Subpackages

DOCUMENTATION FOR LRS AND MRS SUBPACKAGES TO BE ADDED.

## 3.5 MIRI Simulator Software (`miri.simulators`)

**Release** 2.5evxx

**Date** December 18, 2015

### 3.5.1 Introduction

This document describes the implementation details for the MIRI general simulators package (simulators).

### 3.5.2 Modules

General purpose simulator modules may be found in the simulators/lib directory

#### File Searching Utility (`miri.simulators.find_simulator_file`)

#### Description

This module contains a utility which finds a named data file or configuration file which is stored somewhere within the miri.simulators package.

#### Objects

None.

#### Functions

#### Poisson integrator module (`miri.simulators.poisson_integrator`)

#### Description

This module simulates the behaviour of a detector made of electron counting pixels, when the incoming flux is small. It can be reused in any simulation where a particle integrator obeying Poisson statistics is needed.

The imperfect integrator adds zero-point and latency effects.

#### Objects

#### Functions

None.

### 3.5.3 Subpackage - mirimsim

The MIRI imager simulator.

#### MIRI Imager Simulator (`miri.simulators.mirimsim`)

**Release** 2.5evxx

**Date** December 18, 2015

#### LICENCE

**Copyright** Copyright (c) 2010-2014 United Kingdom Astronomy Technology Centre, an establishment of the Science and Technology Facilities Council. All rights reserved.

The SCASim software has been developed by CEA Saclay as part of the JWST/MIRI consortium, which includes the following organisations: Ames Research Center, USA; Netherlands Foundation for Research in Astronomy; CEA Service d'Astrophysique, Saclay, France; Centre Spatial de Liege, Belgium; Consejo Superior de Investigacones Cientificas, Spain; Danish Space Research Institute; Dublin Institute for Advanced Studies, Ireland; EADS Astrium, Ltd., European Space Agency, Netherlands; UK; Institute d'Astrophysique Spatiale, France; Instituto Nacional de Tecnica Aerospacial, Spain; Institute of Astronomy, Zurich, Switzerland; Jet Propulsion Laboratory, USA; Laboratoire d'Astrophysique de Marseille (LAM), France; Lockheed Advanced Technology Center, USA; Max-Planck-Insitut fur Astronomie (MPIA), Heidelberg, Germany; Observatoire de Paris, France; Observatory of Geneva, Switzerland; Paul Scherrer Institut, Switzerland; Physikalishes Institut, Bern, Switzerland; Raytheon Vision Systems, USA; Rutherford Appleton Laboratory (RAL), UK; Space Telescope Science Institute, USA; Toegepast-Natuurwetenschappelijk Ondeszoek (TNOTPD), Netherlands; UK Astronomy Technology Centre (UKATC); University College, London, UK; University of Amsterdam, Netherlands; University of Arizona, USA; University of Cardiff, UK; University of Cologne, Germany; University of Groningen, Netherlands; University of Leicester, UK; University of Leiden, Netherlands; University of Leuven, Belgium; University of Stockholm, Sweden, Utah State University USA.

**Terms and Conditions of Use** This software may be used and copied free of charge only for non-commercial research purposes. All copies of this software must contain this copyright statement and disclaimer. The MIRI consortium must be acknowledged in any publications arising from use of this software. If you make modifications to this software, you must clearly mark the software as having been changed and you must also retain this copyright and disclaimer.

Where this software uses facilities developed by other members of the MIRI consortium (e.g. its use of stsci-python) it is also bound by the licences issued with those facilities (see the LICENCE files contained in other parts of the JWST repository).

**Disclaimer** This software is available "as is", without warranty of any kind, either expressed or implied, including the implied warranties of merchantability and fitness for a specific purpose. By using this software you are assuming all risks and costs. In no event is the Science and Technology Facilities Council or the MIRI consortium liable for any damages or losses that might result from the use of this software.

Any calibration files distributed with this software are for testing purposes and do not describe the status of the MIRI instrument or the quality of the data expected from MIRI.

#### Introduction

MIRI imager simulator.

DOCUMENTATION TO BE INSERTED HERE

**References**

#### Scripts

The mirimsim scripts may be found in the mirimsim/scripts directory.

**mirimsim**

#### Modules

The mirimsim modules may be found in the mirimsim/lib directory.

### 3.5.4 Subpackage - scasim

The MIRI SCA simulator.

#### MIRI Sensor Chip Assembly Simulator (`miri.simulators.scasim`)

**Release** 2.5evxx

**Date** December 18, 2015

#### LICENCE

**Copyright** Copyright (c) 2010-2015 United Kingdom Astronomy Technology Centre, an establishment of the Science and Technology Facilities Council. All rights reserved.

The SCASim software has been developed by the UKATC as part of the JWST/MIRI consortium, which includes the following organisations: Ames Research Center, USA; Netherlands Foundation for Research in Astronomy; CEA Service d'Astrophysique, Saclay, France; Centre Spatial de Liege, Belgium; Consejo Superior de Investigacones Cientificas, Spain; Danish Space Research Institute; Dublin Institute for Advanced Studies, Ireland; EADS Astrium, Ltd., European Space Agency, Netherlands; UK; Institute d'Astrophysique Spatiale, France; Instituto Nacional de Tecnica Aerospacial, Spain; Institute of Astronomy, Zurich, Switzerland; Jet Propulsion Laboratory, USA; Laboratoire d'Astrophysique de Marseille (LAM), France; Lockheed Advanced Technology Center, USA; Max-Planck-Insitut fur Astronomie (MPIA), Heidelberg, Germany; Observatoire de Paris, France; Observatory of Geneva, Switzerland; Paul Scherrer Institut, Switzerland; Physikalishes Institut, Bern, Switzerland; Raytheon Vision Systems, USA; Rutherford Appleton Laboratory (RAL), UK; Space Telescope Science Institute, USA; Toegepast-Natuurwetenschappelijk Ondeszoek (TNOTPD), Netherlands; UK Astronomy Technology Centre (UKATC); University College, London, UK; University of Amsterdam, Netherlands; University of Arizona, USA; University of Cardiff, UK; University of Cologne, Germany; University of Groningen, Netherlands; University of Leicester, UK; University of Leiden, Netherlands; University of Leuven, Belgium; University of Stockholm, Sweden, Utah State University USA.

**Terms and Conditions of Use** This software may be used and copied free of charge only for non-commercial research purposes. All copies of this software must contain this copyright statement and disclaimer. The MIRI consortium must be acknowledged in any publications arising from use of this software. If you make modifications to this software, you must clearly mark the software as having been changed and you must also retain this copyright and disclaimer.

Where this software uses facilities developed by other members of the MIRI consortium (e.g. its use of stsci-python) it is also bound by the licences issued with those facilities (see the LICENCE files contained in other parts of the JWST repository).

**Disclaimer** This software is available "as is", without warranty of any kind, either expressed or implied, including the implied warranties of merchantability and fitness for a specific purpose. By using this software you are assuming all risks and costs. In no event is the Science and Technology Facilities Council or the MIRI consortium liable for any damages or losses that might result from the use of this software.

Any calibration files distributed with this software are for testing purposes and do not describe the status of the MIRI instrument or the quality of the data expected from MIRI.

#### Introduction

This document describes the implementation details for the MIRI Sensor Chip Assembly (SCA) simulator software. It should be read alongside the "MIRI Sensor Chip Assembly (SCA) Simulator User Manual" (1), which describes how to use the software, and the "MIRI Sensor Chip Assembly (SCA) Simulator Software Design Document" (2), which describes the design on which this software implementation is based.

The SCA simulator software is designed to be run from the scripts - e.g. see the description of the scasim script in "Scripts" section below. The scripts are given basic information, such as the input and output file names, the detector temperature and cosmic ray environment to be simulated and the detector readout and subarray modes required. It is also possible to call the SCA simulator directly from Python software - see the description of the simulate_sca_fromdata function within the "sensor_chip_assembly" module. More detailed information about the properties of the SCA (such as how the dark current and read noise changes with temperature) is contained within configuration files - see the "Data" section for details. The "Modules" section describes the detailed contents of the software modules.

The software depends on the tools and utilities in the miritools module.

**References**


1. MIRI Sensor Chip Assembly (SCA) Simulator User Manual, V0.98, 23 November 2011.
2. MIRI Sensor Chip Assembly (SCA) Simulator Software Design Document, V0.91, 27 September 2011.
3. JPL D-25632, MIRI Operations Concept Document (Revision C), Christine Chen, Karl Gordon, Scott Friedman and Margaret Meixner, STScI, 4 March 2010.
4. JPL D-46944, MIRI Flight Focal Plane Module End Item Data Package (FPM EIDP) (edited version), A. Schneider et al., Initial X1, 10 May 2010
5. The JWST MIRI Instrument Concept, Wright et al., Proc. SPIE, 5487, 653, 2004.
6. MIRI DFM 308-04.02, Analysis of the Proton Flux on the MIRI Detector Arrays, M. Ressler, 3 August 2009.
7. JWST-STScI-00198, SM-12, A Library of Simulated Cosmic Ray Events Impacting JWST HgCdTe Detectors, M. Robberto, 9 March 2010. (MIRI update in press)
8. JWST Calibration Software Online Documentation, http://stsdas.stsci.edu/jwst/docs/sphinx/
9. DHAS miri_sloper Online Documentation, http://tiamat.as.arizona.edu/dhas/
10. FM Raw Science Data Definition (Issue 1, Draft B), Tim Grundy, 12 January 2011.
11. MIRI calibration file metadata convention, Jane Morrison, 9 June 2011.
12. MIRI Test Report, Description of Bad Pixel Mask, Jane Morrison, V4, 28 August 2011


### Scripts

The scasim scripts may be found in the scasim/scripts directory.

**scasim**

**make_sca_file**

**make_bad_pixel_mask**

**make_fringe_map**

**make_sca_calibration**

**detector_latency_test**

### Configuration Information

The parameters provided to the scripts, described in the previous section, are used to control those aspects of a MIRI simulation that are expected to change from observation to observation. More detailed control of the simulations is possible by editing the configuration files supplied with the simulator. The files, which may be found in the scasim/lib directory, are:

```
cosmic_ray_properties.py
detector_properties.py
```

These files describe the cosmic ray environment, and the properties of the detector and electronics.

To make a change to one or more of these files, copy them to your current working directory and edit them. As long as the new files are within your current working directory (or a subdirectory within it), there is no need to rebuild the SCAsim software to make the new parameters current. To revert to the original parameters, simple delete or rename the edited files.

SCASim will search the current directory, the MIRI data directories and then the PYTHONPATH directories for these files, and it will report the full path and name of each one used. Make sure SCAsim is using the files you expect.

*NOTE: These configuration files are processed by the Python interpreter as if they were source code. Be careful not to miss out matching quotes or brackets. If there is a problem in interpreting one of these files, you will see a parsing error when SCAsim is executed. The offending file will be named. See the "MIRI Sensor Chip Assembly (SCA) Simulator User Manual" for details.*

#### Shared Modules

The following modules may be found in the top level simulators package, and are shared with other MIRI simulators.

- find_simulator_file
- poisson_integrator

#### Modules

The scasim modules may be found in the scasim/lib directory.

**Sensor chip assembly module (`miri.simulators.scasim.sensor_chip_assembly`)**

**Description** The sensor_chip_assembly module contains the SensorChipAssembly class (the top level class of the MIRI Sensor Chip Assembly simulator) plus associated functions. The top level simulate_sca function runs the SCA simulator; reading a detector illumination input file and writing an output file. The simulate_sca_fromdata function provides a way of running the SCA simulator directly from data, without the need to write a detector illumination file. The overall class hierarchy is:

```
SensorChipAssembly
    CosmicRayEnvironment
        CosmicRay
    MiriIlluminationModel
    MiriBadPixelMaskModel
    MiriExposureModel or ExposureData
       MiriMeasuredModel
    DetectorArray
        ImperfectIntegrator
            PoissonIntegrator
        MiriMeasurement
    ParameterFileManager
    MiriQuantumEfficiency
        MiriFilter
```

The MiriMeasurement, MiriQuantumEfficiency and ParameterFileManager classes are obtained from the miritools package. The software also uses the following configuration files to describe the properties of the MIRI Sensor Chip Assembly and its environment:

```
cosmic_ray_properties.py
detector_properties.py
```

#### Objects

**Functions** Global functions simulate_sca, simulate_sca_list and simulate_pipeline are deprecated. Global function simulate_sca_fromdata has been removed.

#### Detector module (`miri.simulators.scasim.detector`)

**Description** The detector module contains the DetectorArray class, which (together with the amplifier module) simulates the behaviour of a MIRI sensor chip assembly (SCA: IM, LW or SW) with focal plane module (FPM: 106, 104 or 105).

*Note: Other MULTIACCUM detectors (including non-MIRI detectors) can be added by modifying detector_properties.py*

#### Objects

**Functions** None.

**Configuration files** The file detector_properties.py contains configuration information.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

"""

Module detector_properties - Defines the properties of the MIRI SCA
detector.

NOTE: Other JWST detectors can be simulated as long as their
properties can be described by the parameters contained here.
The main differences are the material, pixel size and detector
thickness.

NOTE: The SCASim simulation is as good as the data contained in
these properties files. Initially, the files contained estimated
properties or properties derived from VM measurements. For the best
simulation, the properties derived from the most recent FM
measurements should be used.

Each set of properties is stored in a Python dictionary and the
properties belonging to a particular focal plane module can be
looked up using a dictionary of dictionaries.

Sources:
```

```
(1) JPL D-25632, MIRI Operational Concept Document, 4 March 2010.
(2) JPL D-46944, MIRI Flight Focal Plane Module End Item Data Package
    (FPM EIDP), 10 May 2010.
(3) UA_MIRI_006, MIRI Test Report, Description of Bad Pixel Mask,
    Version 4, 28 August 2011.

:History:

15 Jul 2010: Created
22 Jul 2010: Pixel size and thickness added.
25 Jul 2010: Well depth increased to 100000 (ref 2).
27 Jul 2010: Subarray options added.
01 Aug 2010: Added CHIP_ID and DEFAULTs.
09 Aug 2010: Renamed. QE data file renamed to qe_measurement.txt
16 Aug 2010: Detector properties compiled into a list, since
             MIRI contains three focal plane modules which can behave
             slightly differently. There are now separate dark current
             and QE measurements for each FPM.
24 Aug 2010: QE and dark current measurements now read from FITS files.
             Detector thickness increased to 470 microns.
30 Aug 2010: Locate the configuration files using an absolute file path
             so that scasim may be run from any directory.
03 Sep 2010: Added COSMIC_RAY_LEAKAGE_FRACTION. According to Mike Ressler,
             a small fraction of cosmic ray events can cause charge
             leakage rather than a charge jump.
             Bad pixel maps added.
27 Sep 2010: Python environment for windows verified.
01 Oct 2010: Dropped groups added to readout modes.
12 Oct 2010: Number of frames per group added to readout modes (always 1
             for MIRI but will allow SCAsim to be extended for other JWST
             detectors).
18 Oct 2010: Group gap for GAP modes changed from 4 to 8 to increase the
             exposure time that can be fitted into a reasonable sized
             output file.
11 Nov 2010: DARK_MAP parameters added.
15 Nov 2010: Mistake in CHIP_ID corrected.
22 Nov 2010: Corrections to FPM and SCA IDs in FITS header.
24 Nov 2010: Added test Subarray modes.
15 Dec 2010: SCA_ID values changed from 104,105,106 to 493,494,495
             (as expected by miri_cube)
07 Jan 2011: ID, SCA_ID and DETECTOR values updated to reflect the
             values reported by Tim Grundy (RAL) on 20 Dec 2010.
             Detector properties classified and looked up by SCA_ID
             rather than by FPM_ID.
10 Mar 2011: Subarray modes MASK1065, MASK1550 and SLITLESSPRISM
             corrected to match definitions in OCD Revision C
             (4 March 2010). Subarray modes LRS3, AXIS64, AXIS128 and
             AXIS256 added for test purposes. Subarray modes TEST32
             and TEST64 removed.
25 Mar 2011: PERSISTENCE parameter added, which can be a linear factor
             or a set of polynomial coefficients.
05 Apr 2011: Pixel response function added (not used yet).
21 Sep 2011: The bad pixel masks and dark maps derived from FM tests (3)
             are now used by default.
05 Oct 2011: PERSISTENCE altered for more realistic simulation of
             persistence effects.
             NOTE: Should FRAME_RESETS be increased to simulate the
             post-FM testing adjustment of detector controller parameters?
```

```
            NOTE: Which is the correct detector naming convention:
            Tim Grundy's or  Jane Morrison's?
24 Oct 2011: Modified to use new "filesearching" module, which should
            make it easier for users to substitute their own configuration
            USE_FM_MEASUREMENTS flag removed (too complicated).
            files. Pixel response function removed (never used).
13 Jun 2013: Changed detector and subarray names to match new data model.
25 Oct 2013: Make the primary keyword for determining the detector ID
            'DETECTOR' rather than 'SCA_ID'. Detector properties are
            looked up by detector name rather than sca_id.
24 Feb 2014: Use find_simulator_file function to find auxiliary data files
            rather than searching PYTHONPATH (which takes a long time when
            the software is installed using MIRICLE/ureka).
07 May 2014: Added zeropoint drift, linearity and latency factors.
            Added charge trapping and slope drift factors.
08 May 2014: Charge trapping parameter added.
05 Jun 2014: Removed charge trapping parameters and added slow and fast
            latency parameters. Decay parameters now given as a timescale.
19 Jun 2014: Slow and fast zeropoint drift parameters included.
27 Feb 2015: Subarray parameters brought up to date.
09 Mar 2015: Switch over to CDP-3 bad pixel masks. Well depth adjusted to
            match CDP-3 pixel saturation results (approximately).
11 Mar 2015: Switch over to CDP-3 dark masks. Local detector naming
            convention updated from 493,494,495 to IM,LW,SW.
21 May 2015: Amplifier level removed and gain and read noise defined by
            calibration data products.
06 Aug 2015: Noise calibration factor added.
08 Sep 2015: Made compatible with Python 3.
02 Oct 2015: Removed readout modes which are no longer available.
08 Dec 2015: Correction to subarray parameters to match Mike Ressler's PASP
            paper.

@author: Steven Beard (UKATC)

"""
# For consistency, import the same Python V3 features as the STScI data model.
from __future__ import absolute_import, unicode_literals, division, print_function

from miri.simulators.find_simulator_file import find_simulator_file

#
# MIRI contains three focal plane modules, each of which contains a
# detector with a 1024 x 1024 zone illuminated by the instrument. Each
# focal plane module is identified by a unique Sensor Chip Assembly ID
# and a unique Focal Plane Module ID defined as follows. The SCA ID is
# used in MIRI documentation and the FPM ID is used in the FPM EIDP
# (see email from Tim Grundy, RAL):
#    * For the MIRI imager:
#         SCA 493 containing FPM S/N 106.
#    * For the long wavelength arm of the MIRI MRS:
#         SCA 494 containing FPM S/N 104.
#    * For the short wavelength arm of the MIRI MRS:
#         SCA 495 containing FPM S/N 105.
# Each 1024x1024 pixel detector has 4 extra non-illuminated reference
# columns just off the left and right edges of the illuminated zone.
# In addition, there is a separate bank of non-illuminated reference
# pixels ganged together known as reference outputs. These reference
# outputs are not contiguous with the illuminated zone, but the data
```

```
# is rearranged in level 1 FITS files so these reference outputs appear
# as extra rows on top of the normal detector image.
#
# Note that DARK_CURRENT_FILE describes how the dark current varies
# with detector temperature. DARK_MAP describes how the dark current
# varies over the detector surface (including hot pixels which have
# excessive dark current).
#
# The find_simulator_file function searches for a named file within a
# search path of simulator data files(starting with the current working
# directory) and returns the absolute path.
#
# TODO: Replace with the measured FM/JPL persistence effects.
# TODO: Are there too many fudge factors describing detector drifts?
_sca493 = {}
_sca493['SCA_ID'] = 493             # Numerical SCA ID
_sca493['FPM_ID'] = "FPMSN106"      # Unique FPM ID
_sca493['NAME'] = "Sensor Chip Assembly 493 with Focal Plane Module 106"
_sca493['DETECTOR'] = "MIRIMAGE"    # ASCII detector ID (previously "IM")
_sca493['CHIP'] = 'SiAs'            # Type of detector chip
_sca493['COMMENTS'] = "Describes MIRI FPM S/N 106 detector data with ref pixels"
_sca493['ILLUMINATED_ROWS'] = 1024
_sca493['ILLUMINATED_COLUMNS'] = 1024
_sca493['LEFT_COLUMNS'] = 4    # Reference columns on detector
_sca493['RIGHT_COLUMNS'] = 4   # Reference columns on detector
_sca493['BOTTOM_ROWS'] = 0     # There are no extra rows at the bottom
_sca493['TOP_ROWS'] = 256      # Reference rows in level 1 FITS image
_sca493['PIXEL_SIZE'] = 25.0   # Pixel size in microns
_sca493['THICKNESS'] = 470.0   # Detector thickness in microns
_sca493['WELL_DEPTH'] = 324500 # Well depth in electrons
#_sca493['PERSISTENCE'] = 0.0   # Linear persistence factor (0.0 to 1.0)
_sca493['PERSISTENCE'] = [1.0e-8, 0.03, 0.0]  # Persistence coefficients [2nd,1st,0th]
_sca493['LATENCY_SLOW'] = [1.67e-9, 136000.0] # Slow latency parameters [gain(1/e),decay]
_sca493['LATENCY_FAST'] = [0.002, 300.0]      # Fast latency parameters [gain,decay]
_sca493['ZP_SLOW'] = [45000.0, 0.0084]        # Slow zeropoint drift [const(e),scale(e/s)]
# Fast zeropoint jumps as a function of integration number and flux [[const(e),scale(s)]
_sca493['ZP_FAST'] = [[0.0, -2.917], [0.0, -2.292], [0.0, -2.396], [0.0, -2.408]]
_sca493['LINEARITY'] = [1.0, 0.0]  # Linearity sensitivity coeffs [const,slope]
_sca493['CLOCK_TIME'] = 1.0e-5 # Detector clock time in seconds
_sca493['CLOCK_PER_RESET'] = 3 # Number of clock cycles per reset
_sca493['CLOCK_PER_REF'] = 3   # Extra settling cycles for reference pixels
_sca493['FRAME_RESETS'] = 0    # Extra resets between integrations
_sca493['TARGET_TEMPERATURE'] = 6.7   # Target temperature in K
_sca493['DARK_CURRENT_FILE'] = find_simulator_file('dark_currentIM.fits')
_sca493['QE_FILE'] = find_simulator_file('qe_measurementIM.fits')
# Bad pixel mask and dark map derived from FM data:
_sca493['BAD_PIXEL_MAP'] = find_simulator_file('MIRI_FM_MIRIMAGE_MASK.fits')
_sca493['DARK_MAP'] = find_simulator_file('MIRI_FM_MIRIMAGE_FAST_DARK.fits')
# Original made up bad pixel mask and dark map:
#_sca493['BAD_PIXEL_MAP'] = find_simulator_file('bad_pixelsIM.fits')
#_sca493['DARK_MAP'] = find_simulator_file('dark_mapIM.fits')
_sca493['GAIN'] = find_simulator_file('MIRI_FM_MIRIMAGE_GAIN.fits')
_sca493['READNOISE'] = find_simulator_file('MIRI_FM_MIRIMAGE_SLOW_READNOISE.fits')
_sca493['NOISEFACTOR'] = 0.5 # Noise calibration factor

_sca494 = {}
_sca494['SCA_ID'] = 494             # Numerical SCA ID
_sca494['FPM_ID'] = "FPMSN104"      # Unique FPM ID
```

```
_sca494['NAME'] = "Sensor Chip Assembly 494 with Focal Plane Module 104"
_sca494['DETECTOR'] = "MIRIFULONG"  # ASCII detector ID (previously "LW")
_sca494['CHIP'] = 'SiAs'            # Type of detector chip
_sca494['COMMENTS'] = "Describes MIRI FPM S/N 104 detector data with ref pixels"
_sca494['ILLUMINATED_ROWS'] = 1024
_sca494['ILLUMINATED_COLUMNS'] = 1024
_sca494['LEFT_COLUMNS'] = 4    # Reference columns on detector
_sca494['RIGHT_COLUMNS'] = 4   # Reference columns on detector
_sca494['BOTTOM_ROWS'] = 0     # There are no extra rows at the bottom
_sca494['TOP_ROWS'] = 256      # Reference rows in level 1 FITS image
_sca494['PIXEL_SIZE'] = 25.0   # Pixel size in microns
_sca494['THICKNESS'] = 470.0   # Detector thickness in microns
_sca494['WELL_DEPTH'] = 316250 # Well depth in electrons
#_sca494['PERSISTENCE'] = 0.0   # Linear persistence factor (0.0 to 1.0)
_sca494['PERSISTENCE'] = [1.0e-8, 0.03, 0.0]  # Persistence coefficients [2nd,1st,0th]
_sca494['LATENCY_SLOW'] = [1.67e-9, 136000.0] # Slow latency parameters [gain(1/e),decay(s)]
_sca494['LATENCY_FAST'] = [0.002, 300.0]      # Fast latency parameters [gain,decay(s)]
_sca494['ZP_SLOW'] = [45000.0, 0.0084]        # Slow zeropoint drift [const(e),scale(e/s)]
# Fast zeropoint jumps as a function of integration number and flux [[const(e),scale(s)]
_sca494['ZP_FAST'] = [[0.0, -2.917], [0.0, -2.292], [0.0, -2.396], [0.0, -2.408]]
_sca494['LINEARITY'] = [1.0, 0.0]  # Linearity sensitivity coeffs [const,slope]
_sca494['CLOCK_TIME'] = 1.0e-5 # Detector clock time in seconds
_sca494['CLOCK_PER_RESET'] = 3 # Number of clock cycles per reset
_sca494['CLOCK_PER_REF'] = 3   # Extra settling cycles for reference pixels
_sca494['FRAME_RESETS'] = 0    # Extra resets between integrations
_sca494['TARGET_TEMPERATURE'] = 6.7   # Target temperature in K
_sca494['DARK_CURRENT_FILE'] = find_simulator_file('dark_currentLW.fits')
_sca494['QE_FILE'] = find_simulator_file('qe_measurementLW.fits')
# Bad pixel mask and dark map derived from FM data:
_sca494['BAD_PIXEL_MAP'] = find_simulator_file('MIRI_FM_MIRIFULONG_MASK.fits')
_sca494['DARK_MAP'] = find_simulator_file('MIRI_FM_MIRIFULONG_FAST_DARK.fits')
# Original made up bad pixel mask and dark map:
#_sca494['BAD_PIXEL_MAP'] = find_simulator_file('bad_pixelsLW.fits')
#_sca494['DARK_MAP'] = find_simulator_file('dark_mapLW.fits')
_sca494['GAIN'] = find_simulator_file('MIRI_FM_MIRIFULONG_GAIN.fits')
_sca494['READNOISE'] = find_simulator_file('MIRI_FM_MIRIFULONG_SLOW_READNOISE.fits')
_sca494['NOISEFACTOR'] = 0.5 # Noise calibration factor

_sca495 = {}
_sca495['SCA_ID'] = 495             # Numerical SCA ID
_sca495['FPM_ID'] = "FPMSN105"      # Unique FPM ID
_sca495['NAME'] = "Sensor Chip Assembly 495 with Focal Plane Module 105"
_sca495['DETECTOR'] = "MIRIFUSHORT" # ASCII detector ID (previously "SW")
_sca495['CHIP'] = 'SiAs'            # Type of detector chip
_sca495['COMMENTS'] = "Describes MIRI FPM S/N 105 detector data with ref pixels"
_sca495['ILLUMINATED_ROWS'] = 1024
_sca495['ILLUMINATED_COLUMNS'] = 1024
_sca495['LEFT_COLUMNS'] = 4    # Reference columns on detector
_sca495['RIGHT_COLUMNS'] = 4   # Reference columns on detector
_sca495['BOTTOM_ROWS'] = 0     # There are no extra rows at the bottom
_sca495['TOP_ROWS'] = 256      # Reference rows in level 1 FITS image
_sca495['PIXEL_SIZE'] = 25.0   # Pixel size in microns
_sca495['THICKNESS'] = 470.0   # Detector thickness in microns
_sca495['WELL_DEPTH'] = 321750 # Well depth in electrons
#_sca495['PERSISTENCE'] = 0.0   # Linear persistence factor (0.0 to 1.0)
_sca495['PERSISTENCE'] = [1.0e-8, 0.03, 0.0]  # Persistence coefficients [2nd,1st,0th]
_sca495['LATENCY_SLOW'] = [1.67e-9, 136000.0] # Slow latency parameters [gain(1/e),decay]
_sca495['LATENCY_FAST'] = [0.002, 300.0]      # Fast latency parameters [gain,decay]
```

```
_sca495['ZP_SLOW'] = [45000.0, 0.0084]        # Slow zeropoint drift [const(e),scale(e/s)]
# Fast zeropoint jumps as a function of integration number and flux [[const(e),scale(s)]
_sca495['ZP_FAST'] = [[0.0, -2.917], [0.0, -2.292], [0.0, -2.396], [0.0, -2.408]]
_sca495['LINEARITY'] = [1.0, 0.0]  # Linearity sensitivity coeffs [const,slope]
_sca495['CLOCK_TIME'] = 1.0e-5 # Detector clock time in seconds
_sca495['CLOCK_PER_RESET'] = 3 # Number of clock cycles per reset
_sca495['CLOCK_PER_REF'] = 3   # Extra settling cycles for reference pixels
_sca495['FRAME_RESETS'] = 0    # Extra resets between integrations
_sca495['TARGET_TEMPERATURE'] = 6.7   # Target temperature in K
# The pixel response function is the sensitivity variation across the
# surface of each individual pixel.
_sca495['PIXEL_RESPONSE'] = None      # No pixel response function
_sca495['DARK_CURRENT_FILE'] = find_simulator_file('dark_currentSW.fits')
_sca495['QE_FILE'] = find_simulator_file('qe_measurementSW.fits')
# Bad pixel mask and dark map derived from FM data:
_sca495['BAD_PIXEL_MAP'] = find_simulator_file('MIRI_FM_MIRIFUSHORT_MASK.fits')
_sca495['DARK_MAP'] = find_simulator_file('MIRI_FM_MIRIFUSHORT_FAST_DARK.fits')
# Original made up bad pixel mask and dark map:
#_sca495['BAD_PIXEL_MAP'] = find_simulator_file('bad_pixelsSW.fits')
#_sca495['DARK_MAP'] = find_simulator_file('dark_mapSW.fits')
_sca495['GAIN'] = find_simulator_file('MIRI_FM_MIRIFUSHORT_GAIN.fits')
_sca495['READNOISE'] = find_simulator_file('MIRI_FM_MIRIFUSHORT_SLOW_READNOISE.fits')
_sca495['NOISEFACTOR'] = 0.5 # Noise calibration factor

# Other detector descriptions (e.g. for other JWST instruments) can be
# added here.

# The following constants may be imported by SCA simulator software modules.
#
# Dictionary of known focal plane modules. The detector properties for
# each FPM can be obtained by looking up its unique detector name in this
# dictionary and using the result as another dictionary (i.e. the overall
# data structure is a dictionary of dictionaries).
DETECTORS_DICT = {'MIRIMAGE'    : _sca493,
                  'MIRIFULONG'  : _sca494,
                  'MIRIFUSHORT' : _sca495}

#
# Available readout modes. The tuple contains default values for:
# nsample  - number of A/D samples making up each readout.
# discard  - number of A/D samples discarded before averaging.
# nframes  - number of frames averaged per group.
# groupgap - number of frames dropped between groups.
# ngroups  - default number of readout groups per integration.
# nints    - default number of integrations per exposure.
# avggrps  - number of groups averaged to reduce data rate.
# avgints  - number of integrations averaged to reduce data rate.
#                                      g
#                               n   d  n   r  n       a   a
#                               s   i  f   o  g       v   v
#                               a   s  r   u  r   n   g   g
#                               m   c  a   p  o   i   g   i
#                               p   a  m   g  u   n   r   n
#                               l   r  e   a  p   t   p   t
#                               e   d  s   p  s   s   s   s
READOUT_MODE = {}
READOUT_MODE['SLOW'] =       (10,  2,  1,  0, 10,  1,  1,  1)
READOUT_MODE['FAST'] =       (1,   0,  1,  0,  1, 10,  1,  1)
```

```
READOUT_MODE['FASTINTAVG'] = (1,   0,  1,  0,  1,  4,  1,  4)
READOUT_MODE['FASTGRPAVG'] = (1,   0,  1,  0,  4,  1,  4,  1)
# The following four readout modes were used for MIRI testing only, and they
# will upset the STScI data model if used.
# READOUT_MODE['SLOWINTAVG'] = (10,  2,  1,  0,  1,  4,  1,  4)
# READOUT_MODE['SLOWGRPAVG'] = (10,  2,  1,  0,  4,  1,  4,  1)
# READOUT_MODE['SLOWGRPGAP'] = (10,  2,  4,  8,  4,  1,  1,  1)
# READOUT_MODE['FASTGRPGAP'] = (1,   0,  4,  8,  4,  1,  1,  1)
DEFAULT_READOUT_MODE = 'FAST'


#
# Available subarray options. Row and column numbers start at 1.
# The tuple contains (firstrow, firstcol, subrows, subcolumns)
#
SUBARRAY = {}
SUBARRAY['FULL'] = None
SUBARRAY['MASK1065'] =      (  19,    1,  224,  288)
SUBARRAY['MASK1140'] =      ( 245,    1,  224,  288)
SUBARRAY['MASK1550'] =      ( 467,    1,  224,  288)
SUBARRAY['MASKLYOT'] =      ( 717,    1,  304,  320)
# Old BRIGHTSKY, which touched the left column
#SUBARRAY['BRIGHTSKY'] =     (  37,    1,  512,  968)
SUBARRAY['BRIGHTSKY'] =     (  51,  457,  512,  512)
# Old SUB256, which touched the left column
#SUBARRAY['SUB256'] =        (  37,    1,  256,  668)
SUBARRAY['SUB256'] =        (  51,  413,  256,  256)
SUBARRAY['SUB128'] =        ( 889,    1,  128,  136)
SUBARRAY['SUB64'] =         ( 779,    1,   64,   72)
SUBARRAY['SLITLESSPRISM'] = ( 529,    1,  416,   72)
# The following additional subarray options are used
# for testing and simulation only. The detectors are
# never read out using these modes.
# SUBARRAY['LRS1'] =          (   1,    1, 1024,  420)
# SUBARRAY['LRS2'] =          (   1,  292, 1024,  128)
# SUBARRAY['LRS3'] =          (   1,  292,  512,   64)
# SUBARRAY['AXIS256'] =       ( 384,  388,  256,  256)
# SUBARRAY['AXIS128'] =       ( 448,  452,  128,  128)
# SUBARRAY['AXIS64'] =        ( 480,  484,   64,   64)
# SUBARRAY['TEST64'] =        ( 128,  128,   64,   80)
# SUBARRAY['TEST32'] =        (   8,    8,   32,   32)
# SUBARRAY['RHS256'] =        (   1,  776,  256,  256)
STANDARD_SUBARRAYS = ('FULL', 'MASK1065', 'MASK1140', 'MASK1550', 'MASKLYOT',
                      'BRIGHTSKY', 'SUB256', 'SUB128', 'SUB64', 'SLITLESSPRISM')
DEFAULT_SUBARRAY = 'FULL'

#
# This fraction of cosmic ray events will cause charge leakage (negative jump)
# rather than a charge increase.
# NOTE: The negative jumps are more likely to be caused by a cosmic raye strike
# on the readout electronics rather than a true charge leakage.
#
COSMIC_RAY_LEAKAGE_FRACTION = 0.01

if __name__ == '__main__':
    print( "NOTE: The DetectorProperties module is supposed to be " \
        "imported by another module, not run as a main program." )
    print( "The following detector properties are defined:" )
```

```
    for detid in DETECTORS_DICT:
        print( "DETECTOR %s\n--------------------" % detid )
        detector = DETECTORS_DICT[detid]
        for key in detector:
            print( "%24s = %s" % (key, detector[key]) )
    print( "READOUT_MODE\n------------" )
    for key in READOUT_MODE:
        print( "%24s = %s" % (key, READOUT_MODE[key]) )
    print( "SUBARRAY\n--------" )
    for key in SUBARRAY:
        print( "%24s = %s" % (key, SUBARRAY[key]) )
    print( "Finished." )
```

#### Cosmic ray module (`miri.simulators.scasim.cosmic_ray`)

**Description** This module contains all the classes and functions for simulating cosmic ray events. The module contains two classes. The CosmicRayEnvironment class simulates the cosmic ray environment and is programmed with parameters such as the expected cosmic ray flux and energy distribution. The best simulation can be obtained by reading the library of cosmic ray events published by the STScI (M. Robberto). If the cosmic ray library is not available, the CosmicRayEnvironment class can also generate cosmic ray events at random. Each cosmic ray event is described using the CosmicRay class.

Note that the frequency of cosmic ray events is determined both by the properties of the cosmic rays, such as the expected flux, and by the size of the detector pixels and amplifiers, which determines the target area. The thicker pixels of the MIRI detectors produce longer cosmic ray trails.

##### Objects

##### Functions

**Configuration files** The file cosmic_ray_properties.py contains configuration information.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

"""

Module cosmic_ray_roperties - Defines the properties of the cosmic
rays expected when the JWST is at L2.

Source: Massimo Robberto, JWST-STScI-001928, SM-12, "A library of
simulated cosmic ray events impacting JWST HgCdTe detectors",
2 December 2009.

:History:

25 Jul 2010: Created
02 Aug 2010: Added cosmic ray mode 'NONE' for no cosmic rays.
30 Aug 2010: Locate the configuration files using an absolute file path
             so that scasim may be run from any directory. Check for
             files contained in the JWST cr_sim_ramp_fit data directory.
27 Sep 2010: Python environment for windows verified.
06 Oct 2010: Changed to new cosmic ray library for SiAs detectors.
```

```
24 Oct 2011: Modified to use new "filesearching" module, which should
             make it easier for users to substitute their own configuration
             files (not quite so important for the cosmic ray smiulations).
25 Oct 2011: Corrected bug in energy distribution for NONE mode.
02 Sep 2013: Make more resilient against cr_ramp_fit package not being
             installed.
24 Feb 2014: Use find_cr_library function to find CR library data files
             rather than searching PYTHONPATH (which takes a long time when
             the software is installed using MIRICLE/ureka).
08 Sep 2015: Made compatible with Python 3

@author: Steven Beard (UKATC)

"""
# For consistency, import the same Python V3 features as the STScI data model.
from __future__ import absolute_import, unicode_literals, division, print_function

import os
import warnings

from miri.miritools.filesearching import find_file_in_path

# Get the path to the folder where the cosmic ray library files have been
# installed.
try:
    import cr_sim_ramp_fit
    cr_sim_path = cr_sim_ramp_fit.__path__[0]
    global_cr_data_path = os.path.join(cr_sim_path, 'cr_data_files')
except ImportError:
    warnings.warn("NOTE: cr_sim_ramp_fit package not installed.")
    cr_sim_path = ''
    global_cr_data_path = ''

def find_cr_library( fileprefix, cr_data_path=global_cr_data_path):
    """

    Find the folder containing files with the given named prefix, first
    in a given search path of directories and (failing that) within the
    PYTHONPATH.
    By default, this function will search for a file within the current
    directory the MIRI module  data directories and then try PYTHONPATH.

    This function is used when, rather than there being a single parameter
    file, there are a whole set of parameter files with a common prefix.

    :Parameters:

    fileprefix: str
        A prefix for the files to be located (for example
        ''CR_SiAs_SUNMIN_'', which will cause all files matching
        ''CR_SiAs_SUNMIN_*.*'' to be located).
    cr_data_path: str
        Top level folder where the CR library data files are installed.

    :Returns:

    fileprefixpath: str
        If matching files are found the full path of the folder containing
```

```
        the files plus the prefix is returned (e.g. ''/data/CR_SiAs_SUNMIN_'').
        Returns None if no files are found.

    """
    search_path = os.pathsep + './data'
    if cr_data_path:
        search_path += os.pathsep + cr_data_path

    filename = fileprefix + '*.*'
    firsttry = find_file_in_path( filename, search_path=search_path,
                                  pathsep=os.pathsep, walkdir=True,
                                  path_only=True)
    if firsttry:
        return os.path.join(firsttry, fileprefix)
    else:
        return None


DEFAULT_CR_MODE = 'SOLAR_MIN'

# Cosmic ray flux in number/micron^2/sec incident on the MIRI detectors
# at Solar Minimum, Solar Maximum and during a Solar Flare.
# TODO: Would a GROUND mode be useful? If so, what is the flux and spectrum?
CR_FLUX = {}
CR_FLUX['NONE']        = 0.0
CR_FLUX['SOLAR_MIN']   = 4.8983E-8
CR_FLUX['SOLAR_MAX']   = 1.7783E-8
CR_FLUX['SOLAR_FLARE'] = 3.04683E-5
# The following parameter can be used to adjust the cosmic ray flux
# for testing and fine tuning.
CR_FLUX_MULTIPLIER = 1.0

#
# NOTE: There are three sets of library files, suitable for detectors
# with limiting wavelengths of 1.7, 2.5 and 5.5 microns. The 5.5 files
# are chosen as being the closest simulation to the MIRI detectors
# (although these are thicker).
# TODO: If SCASim is to be extended, choose CR library file by detector type.
#
CR_LIBRARY_FILES = {}
CR_LIBRARY_FILES['NONE'] = None
# Library files for the NIRCAM and NIRSPEC detectors (thickness 5.5 microns).
#CR_LIBRARY_FILES['SOLAR_MIN']   = find_cr_library('CRs_MCD5.5_SUNMIN')
#CR_LIBRARY_FILES['SOLAR_MAX']   = find_cr_library('CRs_MCD5.5_SUNMAX')
#CR_LIBRARY_FILES['SOLAR_FLARE'] = find_cr_library('CRs_MCD5.5_FLARES')
# Library files for the MIRI detectors.
if cr_sim_path:
    CR_LIBRARY_FILES['SOLAR_MIN']   = find_cr_library('CRs_SiAs_SUNMIN_')
    CR_LIBRARY_FILES['SOLAR_MAX']   = find_cr_library('CRs_SiAs_SUNMAX_')
    CR_LIBRARY_FILES['SOLAR_FLARE'] = find_cr_library('CRs_SiAs_FLARES_')
else:
    CR_LIBRARY_FILES['SOLAR_MIN']   = None
    CR_LIBRARY_FILES['SOLAR_MAX']   = None
    CR_LIBRARY_FILES['SOLAR_FLARE'] = None
CR_LIBRARY_FILES['MIN'] = 0
CR_LIBRARY_FILES['MAX'] = 9
# The following parameter can be used to adjust the number of electrons
# released by any given cosmic ray for testing and fine tuning.
```

```
CR_ELECTRON_MULTIPLIER = 1.0

CR_NUCLEONS = ('H', 'He', 'C', 'N', 'O', 'Fe', '??')

# A map of the leakage of CR flux from a detector pixel into adjacent
# pixels by capacitative coupling.
# TODO: This is a general detector effect, rather than only applying to cosmic ray hits. It also caus
CR_COUPLING = ((0.0,   0.015, 0.0), \
               (0.015, 0.94,  0.015), \
               (0.0,   0.015, 0.0))

#
# If the library files are not available, this set of electron distributions
# can be used to generate random cosmic ray events independently (although
# not as realistically). The lists contain (energy,probability), where the
# probabilities are incremental and normalised to a peak of 1.0. (They are
# converted to a cumulative distribution internally by the software.)
#
CR_ELECTRONS = {}
CR_ELECTRONS['NONE'] = \
    (
     (0.0,   1.0),
     (1.0e1, 0.0),
     (1.0e2, 0.0),
     (1.0e3, 0.0),
     (1.0e4, 0.0),
     (1.0e5, 0.0),
     (1.0e6, 0.0),
     (1.0e7, 0.0),
     )
CR_ELECTRONS['SOLAR_MIN'] = \
    (
     (1.0e1, 0.0),
     (1.8e1, 0.0),
     (3.2e1, 0.0),
     (5.6e1, 0.02),
     (1.0e2, 0.05),
     (1.8e2, 0.05),
     (3.2e2, 0.05),
     (5.6e2, 0.07),
     (1.0e3, 0.08),
     (1.8e3, 0.10),
     (3.2e3, 0.20),
     (5.6e3, 0.3),
     (1.0e4, 0.5),
     (1.8e4, 1.0),
     (3.2e4, 0.6),
     (5.6e4, 0.15),
     (1.0e5, 0.08),
     (1.8e5, 0.05),
     (3.2e5, 0.03),
     (5.6e5, 0.01),
     (1.0e6, 0.01),
     (1.8e6, 0.003),
     (3.2e6, 0.001),
     (5.6e6, 0.0),
     (1.0e7, 0.0),
     (1.8e7, 0.0),
```

```
     (3.2e7, 0.0),
     (5.6e7, 0.0),
     )
CR_ELECTRONS['SOLAR_MAX'] = \
    (
     (1.0e1, 0.0),
     (1.8e1, 0.0),
     (3.2e1, 0.0),
     (5.6e1, 0.01),
     (1.0e2, 0.02),
     (1.8e2, 0.02),
     (3.2e2, 0.02),
     (5.6e2, 0.03),
     (1.0e3, 0.05),
     (1.8e3, 0.07),
     (3.2e3, 0.15),
     (5.6e3, 0.4),
     (1.0e4, 1.0),
     (1.8e4, 0.25),
     (3.2e4, 0.1),
     (5.6e4, 0.05),
     (1.0e5, 0.05),
     (1.8e5, 0.05),
     (3.2e5, 0.05),
     (5.6e5, 0.01),
     (1.0e6, 0.01),
     (1.8e6, 0.005),
     (3.2e6, 0.001),
     (5.6e6, 0.0),
     (1.0e7, 0.0),
     (1.8e7, 0.0),
     (3.2e7, 0.0),
     (5.6e7, 0.0),
     )
CR_ELECTRONS['SOLAR_FLARE'] = \
    (
     (1.0e2, 0.0),
     (1.8e2, 0.001),
     (3.2e2, 0.001),
     (5.6e2, 0.03),
     (1.0e3, 0.05),
     (1.8e3, 0.06),
     (3.2e3, 0.06),
     (5.6e3, 0.07),
     (1.0e4, 0.1),
     (1.8e4, 0.12),
     (3.2e4, 0.25),
     (5.6e4, 0.6),
     (1.0e5, 1.0),
     (1.8e5, 0.6),
     (3.2e5, 0.5),
     (5.6e5, 0.37),
     (1.0e6, 0.18),
     (1.8e6, 0.07),
     (3.2e6, 0.04),
     (5.6e6, 0.03),
     (1.0e7, 0.02),
     (1.8e7, 0.01),
```

```
      (3.2e7, 0.0),
      (5.6e7, 0.0),
      )


if __name__ == '__main__':
    print( "NOTE: The CosmicRayProperties module is supposed to be " \
        "imported by another module, not run as a main program." )
    print( "The following cosmic ray properties are defined:" )
    print( "CR_FLUX\n-------" )
    for key in CR_FLUX:
        print( "%16s = %s" % (key, CR_FLUX[key]) )
    print( "CR_LIBRARY_FILES\n----------------" )
    for key in CR_LIBRARY_FILES:
        print( "%16s = %s" % (key, CR_LIBRARY_FILES[key]) )
    print( "CR_NUCLEONS\n--------------" )
    print( CR_NUCLEONS )
    print( "CR_COUPLING\n--------------" )
    for row in CR_COUPLING:
        print( "(%.3f, %.3f, %.3f)" % row )
    print( "CR_ELECTRONS\n--------------" )
    for key in CR_ELECTRONS:
        for row in CR_ELECTRONS[key]:
            str = key
            str += ": Relative probability of %.0f electrons is %f" % row
            print( str )
    print( "Finished." )
```

### Exposure data module (`miri.simulators.scasim.exposure_data`)

**Description** This module manages the exposure data generated by the SCA simulator and contains functions for writing that exposure data to a FITS file in FITSWriter or DMS 'level 1' format. The ExposureData class manages the data generated by one MIRI exposure. The exposure data contains each of the 2-D images read out from the detector for each readout group and each integration belonging to an exposure.

*NOTE: This module exists to support legacy data formats only. The module miri.miritools.dataproduct.miri_exposure_model contains the more recent MiriExposureModel class, which uses the jwst_lib data model.*

#### Objects

#### Functions

**File Format** The exposure data is written either to an STScI data model object (the default), a 'FITSWriter' file, or a 'level 1' FITS file.

In 'level 1' format, the exposure data is written to an extension HDU named 'SCI'. In 'FITSWriter' format that same data is written to the primary HDU. There are two alternative formats for the data array:

- A 3-D data cube containing a stack of 2-D detector readout images for each integration and group. The default.
- a 4-D hypercube containing a stack of 3-D cubes for each integration; and each cube containing a stack of 2-D images for each group.

#### Measurement Data

`miri.scasim` comes with configuration data describing the cosmic ray, detector and amplifier properties, plus dark current, read noise and quantum efficiency measurements. The measurement data can be provided in ASCII or FITS format files. These files are normally stored in the scasim/data directory, but if SCASim detects files with the same name stored in the current working directory it will use those in preference.. The ASCII files are useful for storing measurement data cut and pasted from elsewhere, whereas the FITS files are more useful because they can store ancilliary information (such as measurement comments) in their header. The 'miritools.make_measurements_fits'and 'make_qe_fits' scripts may be used to convert the ASCII format data into FITS. The make_bad_pixel_mask script can be used to create a tailored bad pixel mask in FITS format. Test calibration files (including bad pixel maps and dark maps) can be made with the make_sca_calibration script. Some examples are given below:

*NOTE: SCAsim is now configured by default to use the bad pixel maps and dark current maps derived from FM data by DHAS (e.g. MIRI_FM_BadPixelMask_XXX.fits and MIRI_FM_DarkMap_XXX.fits, where XXX is IM, SW or LW).*

**./data/amplifiers/read_noiseXXX.txt** An ASCII file containing an amplifier read noise measurement for Sensor Chip Assembly XXX (e.g. IM, SW or LW). The equivalent FITS file has the same name plus a '.fits' extension.

**./data/detector/bad_pixelsXXX.fits** A FITS file containing a bad pixel mask (for Sensor Chip Assembly XXX).

**./data/detector/dark_currentXXX.txt** An ASCII file containing a detector dark current measurement (for Sensor Chip Assembly XXX). The equivalent FITS file has the same name plus a '.fits' extension.

**./data/detector/dark_mapXXX.fits** A FITS file containing a dark current variation map (for Sensor Chip Assembly XXX).

**./data/detector/qe_measurementXXX.txt** An ASCII file containing a detector quantum efficiency measurement (for Sensor Chip Assembly XXX). The equivalent FITS file has the same name plus a '.fits' extension.

**./data/SCATestInput80x64.fits** An example SCA-format FITS file containing an 80x64 pixel detector illumination image.

**Read Noise Measurement Example (ASCII format)**

**Dark Current Measurement Example (ASCII format)**

**Quantum Efficiency Measurement Example (ASCII format)**

**Bad Pixel Specification Example (ASCII format)**

```
#
# An example ASCII file defining a bad pixel mask.
# The mask will be 32 x 32 pixels in size and
# contain bad pixels looking like the word "bad".
# Rows and columns are numbered from 0 to 31.
#
31 31  0
 2  2  1
 2  3  1
 2  4  1
 3  2  1
 3  4  1
 3  5  1
 4  2  1
 4  5  1
 5  2  1
```

```
5  3  1
5  4  1
5  5  1
6  2  1
6  4  1
6  5  1
7  2  1
7  4  1
8  2  1
8  3  1
8  4  1

2  9  1
2 15  1
3  9  1
3 10  1
3 14  1
3 15  1
4 10  1
4 14  1
5 10  1
5 11  1
5 12  1
5 13  1
5 14  1
6 11  1
6 13  1
7 11  1
7 13  1
8 12  1

2 19  1
2 20  1
2 21  1
3 19  1
3 22  1
4 19  1
4 23  1
5 19  1
5 23  1
6 19  1
6 23  1
7 19  1
7 22  1
8 19  1
8 20  1
8 21  1
```

## 3.6 MIRI STScI Pipeline Support (`miri.stpipeline`)

**Release** 2.5evxx

**Date** December 18, 2015

INSERT DOCUMENTATION FOR MIRI STSCI PIPELINE SUPPORT PACKAGE HERE.

CHAPTER

FOUR

# APPENDIX - INSTALLATION UTILITIES

## 4.1 Installation utilities (`miri.setup_utils`)

### 4.1.1 Description

This module contains general purpose functions for installing and checking the MIRI software.

### 4.1.2 Functions

### 4.1.3 Scripts

The script miri_installation_check.py can be run to check the installation of the MIRI software and report the version numbers of the Python libraries installed.

## m


miri, 45
miri.miritools, 8
miri.miritools.dataproduct, 9
miri.miritools.dataproduct.compatibility, 19
miri.miritools.dataproduct.dqflags, 20
miri.miritools.dataproduct.miri_badpixel_model, 10
miri.miritools.dataproduct.miri_dark_reference_model, 10
miri.miritools.dataproduct.miri_distortion_models, 11
miri.miritools.dataproduct.miri_droop_model, 11
miri.miritools.dataproduct.miri_exposure_model, 18
miri.miritools.dataproduct.miri_flatfield_model, 12
miri.miritools.dataproduct.miri_fluxconversion_models, 12
miri.miritools.dataproduct.miri_gain_model, 13
miri.miritools.dataproduct.miri_illumination_model, 18
miri.miritools.dataproduct.miri_jump_model, 13
miri.miritools.dataproduct.miri_lastframe_model, 13
miri.miritools.dataproduct.miri_latent_model, 14
miri.miritools.dataproduct.miri_linearity_model, 14
miri.miritools.dataproduct.miri_measured_model, 9
miri.miritools.dataproduct.miri_model_base, 9
miri.miritools.dataproduct.miri_photometric_models, 15
miri.miritools.dataproduct.miri_pixel_saturation_model, 15
miri.miritools.dataproduct.miri_psf_models, 16
miri.miritools.dataproduct.miri_reset_model, 16
miri.miritools.dataproduct.miri_straylight_model, 16
miri.miritools.dataproduct.miri_telescope_emission_, 17
miri.miritools.dataproduct.miri_transmission_correc, 17
miri.miritools.dataproduct.miri_wavelength_correcti, 18
miri.miritools.dataproduct.operations, 19
miri.miritools.dataproduct.plotting, 20
miri.miritools.filesearching, 8
miri.miritools.measurements, 20
miri.miritools.measurements.miri_filters, 21
miri.miritools.measurements.miri_measurement, 22
miri.miritools.miriplot, 8
miri.setup_utils, 45
miri.simulators, 24
miri.simulators.find_simulator_file, 24
miri.simulators.mirimsim, 26
miri.simulators.poisson_integrator, 25
miri.simulators.scasim, 29
miri.simulators.scasim.cosmic_ray, 37
miri.simulators.scasim.detector, 30
miri.simulators.scasim.exposure_data, 42
miri.simulators.scasim.sensor_chip_assembly, 29
miri.spectroscopy, 23
miri.spectroscopy.spectraldata, 23

## M

miri (module), 1, 45
miri.miritools (module), 8
miri.miritools.dataproduct (module), 9
miri.miritools.dataproduct.compatibility (module), 19
miri.miritools.dataproduct.dqflags (module), 20
miri.miritools.dataproduct.miri_badpixel_model (module), 10
miri.miritools.dataproduct.miri_dark_reference_model (module), 10
miri.miritools.dataproduct.miri_distortion_models (module), 11
miri.miritools.dataproduct.miri_droop_model (module), 11
miri.miritools.dataproduct.miri_exposure_model (module), 18
miri.miritools.dataproduct.miri_flatfield_model (module), 12
miri.miritools.dataproduct.miri_fluxconversion_models (module), 12
miri.miritools.dataproduct.miri_gain_model (module), 13
miri.miritools.dataproduct.miri_illumination_model (module), 18
miri.miritools.dataproduct.miri_jump_model (module), 13
miri.miritools.dataproduct.miri_lastframe_model (module), 13
miri.miritools.dataproduct.miri_latent_model (module), 14
miri.miritools.dataproduct.miri_linearity_model (module), 14
miri.miritools.dataproduct.miri_measured_model (module), 9
miri.miritools.dataproduct.miri_model_base (module), 9
miri.miritools.dataproduct.miri_photometric_models (module), 15
miri.miritools.dataproduct.miri_pixel_saturation_model (module), 15
miri.miritools.dataproduct.miri_psf_models (module), 16
miri.miritools.dataproduct.miri_reset_model (module), 16
miri.miritools.dataproduct.miri_straylight_model (module), 16
miri.miritools.dataproduct.miri_telescope_emission_model (module), 17
miri.miritools.dataproduct.miri_transmission_correction_model (module), 17
miri.miritools.dataproduct.miri_wavelength_correction_model (module), 18
miri.miritools.dataproduct.operations (module), 19
miri.miritools.dataproduct.plotting (module), 20
miri.miritools.filesearching (module), 8
miri.miritools.measurements (module), 20
miri.miritools.measurements.miri_filters (module), 21
miri.miritools.measurements.miri_measurement (module), 22
miri.miritools.miriplot (module), 8
miri.setup_utils (module), 45
miri.simulators (module), 24, 29
miri.simulators.find_simulator_file (module), 24
miri.simulators.mirimsim (module), 26
miri.simulators.poisson_integrator (module), 25
miri.simulators.scasim (module), 29
miri.simulators.scasim.cosmic_ray (module), 37
miri.simulators.scasim.detector (module), 30
miri.simulators.scasim.exposure_data (module), 42
miri.simulators.scasim.sensor_chip_assembly (module), 29
miri.spectroscopy (module), 23
miri.spectroscopy.spectraldata (module), 23